GFilter: A General Gram Filter for String Similarity Search

Haoji Hu, Kai Zheng, *Member, IEEE*, Xiaoling Wang, *Member, IEEE*, and Aoying Zhou, *Member, IEEE*

Abstract—Numerous applications such as data integration, protein detection, and article copy detection share a similar core problem: given a string as the query, how to efficiently find all the similar answers from a large scale string collection. Many existing methods adopt a prefix-filter-based framework to solve this problem, and a number of recent works aim to use advanced filters to improve the overall search performance. In this paper, we propose a gram-based framework to achieve near maximum filter performance. The main idea is to judiciously choose the high-quality grams as the prefix of query according to their estimated ability to filter candidates. As this selection process is proved to be NP-hard problem, we give a cost model to measure the filter ability of grams and develop efficient heuristic algorithms to find high-quality grams. Extensive experiments on real datasets demonstrate the superiority of the proposed framework in comparison with the state-of-art approaches.

Index Terms—Data integration, similarity search, gram-based framework

1 Introduction

Similarity search has attracted considerable attention from database community recently, due to its broad range of applications in data cleaning, near-duplicate detection, natural language processing and so on. For example, data records that represent the identical real-world entities may have minor differences in their representations when they are merged from different data sources. In this case the similar records need to be detected and correlated. A common task in protein detection is to find all protein sequences with base similar sequences to a given template. Modern search engines often apply similarity search to suggest relevant query keywords to fix the typos in user queries. All such problems can be addressed by similarity search: given a collection of strings and a query string, find all strings similar to the query string. There are many distance measures to calculate the similarity, such as Jaccard, cosine and edit distance. Because edit distance is widely used, in this paper we adopt edit distance.

Existing literatures usually adopt a gram-based filter-and-verification framework for string similarity search. After the query string is decomposed into grams, an efficient and critical type of filter can utilize the grams and inverted index (from grams to strings) of the string collection to generate candidates. Other advanced filters can be also used after this elementary type of filter. Then in verification step, the similarity function is evaluated for the surviving candidates to produce the final results. Prefix filter [5] is a dominant technique in the gram-based similarity search problem. The intuition is that if two strings are similar they must share some common parts. Counter filter [1] is another important filter, which uses all the grams of a string as the prefix. Hence counter filter can also be regarded as a special case of general prefix filtering. Traditional prefix filters focus on minimizing the filtering cost, whereas counter filter aims at reducing the verification cost. Existing frameworks usually use only one of such filters, which misses the chance to seek a better filter case by case. Thus, a dilemma of choosing filters is encountered.

Generally, filtering time cost and verification time cost are a tradeoff in filter-and-verification framework. Actually, traditional prefix filter can be also generalized by extending the prefix. In recent study, Wang et al. [7] demonstrate that adding extra grams can decrease the candidate size by sacrificing a little estimation and filtering time. They formulate a tradeoff between filtering cost and verification cost, which is the basic principle used in the prefix-extending algorithm. However, our analysis shows this scheme may not be high quality combination of grams for a given query.

Now two questions arise naturally: 1) *does an optimal combination of grams used by filters for a given query exist, which can achieve the least total cost*? And if so, 2) *how can we find these high quality gram sets efficiently*? This paper presents our findings when trying to answer the two questions. First a general gram filter is proposed. We formulate the instantiation of general gram filter as an optimization problem. This problem is NP-hard by any exact algorithm. Then we propose efficient algorithm to find a sub-optimum solution. Our proposed method consists of two steps: (a) choosing high quality grams as the base query-gram set; (b) extending the prefix by choosing more grams until no benefit can be gained from the new gram. A series of heuristics are proposed to achieve efficiency and effectiveness of our

- *H. Hu, X. Wang, and A. Zhou are with the Shanghai Key Laboratory of Trustworthy Computing, Software Engineering Institute, East China Normal University, Shanghai, China. E-mail: hjhu@ecnu.cn, {xlwang, ayzhou}@sei.ecnu.edu.cn.*
- *K. Zheng is with the School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD 4072, Australia E-mail: kevinz@itee.uq.edu.au.*

Manuscript received 22 Jan. 2014; revised 29 July 2014; accepted 1 Aug. 2014. Date of publication 19 Aug. 2014; date of current version 3 Mar. 2015. Recommended for acceptance by C.-Y. Chan. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TKDE.2014.2349914

algorithms. To sum up, we make the following major contributions in this paper:

1) We develop a general gram filter. This generalization provides a chance to select optimized combination of grams from q-gram set of a query. Theoretical analysis is conducted on the complexity of the problem, which is NP-hard.
2) We present a choose-and-extend framework to efficiently find the high quality grams in the query process. Under this framework, different strategies can be extended.
3) We propose a new measure of analyzing the overlap between inverted lists. This is the key of synthetic criterion to help generate better ordering for the grams of query. With the help of synthetic criterion, grams can be selected efficiently and effectively.
4) Extensive experiments based on real data sets are conducted, which shows that our method can achieve better performance than existing state-of-art methods, especially when edit distance increases.

The rest of this paper is organized as follows. The problem is formally defined in Section 2. We analyze the former work and give the motivation in Section 3. Section 4 introduces the general gram filter and provides theoretical analysis. A choose-and-extend framework to obtain high quality query grams is presented in Section 5. A new rank based method is proposed in Section 6. The experimental results are explained in Section 7. A brief review of the related work is introduced in Section 8, followed by conclusion of this paper in Section 9.

2 Preliminaries

2.1 Problem Definition

Given two strings r and s, a distance function $d(r, s)$ is defined to quantify the difference between r and s. Two strings are considered similar if $d(r, s) \leq \tau$. Despite the large number of distance functions proposed, we adopt the edit distance in this paper, as it is the most widely-used distance function in string similarity search. The problem of similarity search can be formally defined as follows.

Definition 1 (Similarity Search). *Given a string collection $\mathcal{S}$, a query string Q and a distance threshold τ, the problem of* similarity search *is to find all the strings $S \in \mathcal{S}$, such that $d(Q, S) \leq \tau$.*

Obviously it is not efficient to directly calculate the distance between each string in $\mathcal{S}$ and the query, due to the large size of string collection. Therefore many indexing and searching techniques have been proposed to improve the efficiency of the query processing, among which the gram-based approach is the most popular one.

In the sequel, we will review the framework of this method.

2.2 Gram Based Filter-and-Verification Framework

Gram based filter-and-verification framework is widely studied in many papers [1], [5], [6], [7], [8], [9], [4].

Definition 2 (q-gram). *Given a string s and a positive integer q, q-grams of s are obtained by sliding a window of length q over s. If the q-gram at the end of string are fewer than q, we append $q - (|s| \bmod q)$ '\$' as complements, if necessary.[1] The set containing all the q-grams of s is q-gram set of s, denote it by $G_q(s)$.*

Definition 3 (q-chunk). *Given a string s and a positive integer q, q-chunks are just substrings of length q that starts at $1 + i \cdot q$ positions in the string. The set contains all the q-chunks of s is q-chunk set of s, denote it by $C_q(s)$.*

For example, consider string $abcdef$ and $q = 3$, the 3-gram set and 3-chunk set are {abc, bcd, cde, def, $ef\$$, $f\$\$$} and {abc, def} respectively. Note that q-chunk set is a subset of q-gram set. The idea of using chunk in [6] can further improve the query process efficiency, since it accesses less inverted lists.

Gram-based approaches usually consist of three steps:

- *Indexing step.* In this step, an inverted index is built on the string collection, which takes all the grams as the keys and the strings that contain the gram as the values.
- *Candidate generation step.* This step probes the inverted index by using the grams in the query string and in the meantime counts the occurrences of the strings in the chosen lists. A string will be a candidate if its occurrences are above a lower bound. Some filter technologies such as length filter [1] can be used in this step to reduce the number of candidates.
- *Verification step.* The actual distance between the query and candidates is calculated and a candidate will be added to the final results, if the distance is no larger than the given threshold τ.

The last two steps correspond to the filter-and-verification framework, which is widely adopted in most gram-based query process. In state-of-the-art approaches, the most two important types of filters—counter filter and prefix filter, are widely used in the candidates generation step to decide the grams to be used. The basic idea of these filters is that if two strings are similar, they must share some common grams.

Counter filter. Given the distance threshold τ, if $d(s_1, s_2) \leq \tau$, then they share at least the following number of common grams $\max(|G_q(s_1)|, |G_q(s_2)|) - \tau \cdot q$ in $G_q(s_1)$ and $G_q(s_2)$, denoted by LB, where q represents the gram length and $G_q(s_1)$ represents the q-gram set of string s_1. If only the query string s_1 is considered, the lower bound of the number of common grams LB is $|G_q(s_1)| - \tau \cdot q$, where $|G_q(s_1)|$ is the number of q-grams. If we partition the query string into non-overlap segments with fixed length, called q-chunk, the common chunks lower bound [6] becomes $|C_q(s_1)| - \tau$, where $|C_q(s_1)|$ is the number of q-chunks.

Consider an example, as illustrated in Fig. 1, a query string *abcde*, a data string *abde*, which is indexed, and $\tau = 1$. By decomposing both of them into 2-gram, {$ab, bc, cd, de, e\$$} and {$ab, bd, de, e\$$}, the least common grams lower bound

1. If the query is decomposed by q-chunk and collection strings are decomposed by q-gram, \$ is necessary. In our paper, only query string can be decomposed by either q-gram or q-chunk, string collection uses q-gram.

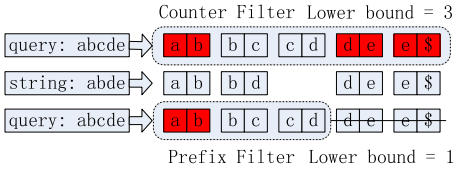


Fig. 1. Example of counter filter and prefix filter.

is 3. Obviously, they share 3 common grams $\{ab, de, e\$\}$. So query string *abcde* can access *abde* through inverted list of grams $\{ab, de, e\$\}$. Since *abde* occurs in 3 gram lists and $LB = 3$, it is a candidate similar to *abcde*. If we partition the query string into chunk set $\{ab, cd, e\$\}$, the analysis is similar.

Prefix filter. Given a global order ∂ for all the appeared grams,[2] and the distance threshold τ, if $d(s_1, s_2) \leq \tau$, then the $(\tau \cdot q + 1)$-prefix of s_1 and s_2 must share at least one gram. l-prefix represents the first l grams of q-gram set sorted by ∂. For q-chunk, we just need to consider the $(\tau + 1)$-prefix for query s_1. The common grams lower bound LB is always 1.

A global order can guarantee correctness of removing some grams to be indexed. However, this is based on an assumption knowing maximum τ. The assumption can't be usually held in query situation, since τ is a part of query which can't be known in advance.

Continuing with the previous example shown in Fig. 1, and assuming the grams are already sorted by global order, we only need to consider the 3-prefix of both strings, i.e., $\{ab, bc, cd\}$ and $\{ab, bd, de\}$. Since query and data string are similar, they must share at least 1 gram in prefixes. Obviously, gram ab occurs in both prefixes.

In order to distinguish with q-gram set, we use *query-gram set* to denote the grams used to access inverted index, the actual grams used as filter, though counter filter and prefix filter use different grams to access the inverted index. Consider string $abcde$, whose 2-gram set is $\{ab, bc, cd, de, e\$\}$, and $\tau = 1$, prefix filter just uses gram set $\{ab, bc, cd\}$ as query-gram set while counter filter uses $\{ab, bc, cd, de, e\$\}$.

In the following, inadequacies of existing framework is illustrated first. Then to solve this problems, a general gram filter is introduced.

3 Motivation

3.1 Dilemma of Choosing Filters

Traditionally, the filtering step will choose either counter filter or prefix filter as the base filter to generate candidates. In the next we will show an example that, the impact of different filters on the search performance may vary depending on the queries. In other words, it is difficult to choose the suitable filter before the query is given.

Consider the inverted index[3] in Fig. 2 and a query string $bcbdcdef$, where the 2-chunk set of the query is $\{bc, bd, cd, ef\}$. Gram bd will never be used since there is no entry for it in the inverted index, which means the list length is 0. If τ is equal to 1, a prefix filter just needs to access

2. Grams are sorted in ascending based on list length of gram.

3. The second element of ef is 3 in this section. In Section 6, it is 4. List length increases from top to down.

gram		string id
ab	→	1,3,5,7
bc	→	1,3,9,11
cd	→	1,3,7,9,11
de	→	1,4,5,10,15
ef	→	1,3(4),9,10,16
fg	→	1,2,7,8,13,14

Fig. 2. Inverted lists of string id.

$\tau + 1 = 2$ inverted list, including bc and bd, resulting in only four elements accessed and four candidates generated, i.e., $\{1, 3, 9, 11\}$. However, a counter filter will access more inverted lists of cd and ef, where 10 more elements will be accessed and the candidate set is $\{1, 3, 9\}$, in which each element appears in at least $LB = |C_2(s)| - \tau = 4 - 1 = 3$ lists. Although counter filter reduces the candidate size by 1, a prefix filter will benefit from less accessed elements.

However, the situation will be completely different, if we change τ from 1 to 2 and use another query string $abbcdefg$, in which case the 2-chunk set of the query is $\{ab, bc, de, fg\}$. Now a prefix filter will access the inverted lists of ab, bc, and de that probe 13 elements and generate the candidates $\{1, 3, 4, 5, 7, 9, 10, 11, 15\}$, whereas a counter filter just needs to access six more elements in gram list fg. But the new candidate set is $\{1, 3, 4, 9, 10\}$, where four more candidates are removed. In this case, the counter filter will obtain a much smaller candidate set, thus reducing the verification cost significantly. Generally speaking, prefix filters need to access less elements at the price of large candidate set, while counter filters usually produce less candidates by probing more inverted lists.

Since the query string and τ can never be known in advance, it is hard to decide which filter should be adopted in the similarity search before query process. The essential reason for this dilemma is that, *both prefix filter and counter filter only consider a single factor: prefix filter only aims at accessing the least number of elements and counter filter just focuses on generating the least number of candidates.*

3.2 Problem of Predefined Gram Sort

Existing frameworks select grams just considering the gram inverted list length. Traditional prefix filter only choose $D + 1$ grams as filter scheme, where D is the number of *destroyed gram*.[4] Adaptive-Search algorithm [7] adds extra grams to further improve the performance. Notwithstanding, all the existing work sort the $G_q(Q)$, defined in advance, based on the length of gram inverted list.

In Adaptive-Search algorithm, more grams can be appended to the prefix while increasing the lower bound of common grams between a candidate pair. However, when a gram of a query chosen by Adaptive-Search algorithm is contained by most of the candidate strings generated based on other selected grams, it should be discarded since there are few benefits of reducing candidate size. In other words,

4. An edit operation can destroy at most q grams under q-gram segmentation and destroy at most 1 grams under q-chunk segmentation. Destroyed grams represent the maximal number of grams that can be destroyed by a given edit distance. D is $q * \tau$ and τ for q-gram and q-chunk respectively.

their approach does not try to optimize the selectivity of the whole gram set.

Consider Fig. 2 as an example. All the grams are sorted in ascending order based on corresponding inverted lists length from top to bottom. Assuming destroyed gram $D = 1$, grams ab and bc have been selected as base prefix scheme for given query. All the strings occurring at least once in ab and bc lists' will enter candidate set. Then candidate set $\{1, 3, 5, 7, 9, 11\}$ is generated. If another new gram cd is added to prefix, the new prefix generates candidates $\{1, 3, 7, 9, 11\}$. Only one string is pruned. However, if we add gram de instead of cd, the new prefix generates candidates $\{1, 3, 5\}$. Two more strings can be pruned. Sort based upon list length is not appropriate. Distinguished with all existing methods, in this paper we take a novel perspective to try to optimize the total time cost of similarity search by strategically selecting a gram set for a query that is expected to achieve a better balance between filtering and verification costs. Specifically, we aim to build a gram set that has small size but great capability to prune candidates, so that both filtering and verification cost can be small.

In fact, different combination of grams will lead to different filters. In the following, we will propose a general filter that treats all the above-mentioned filters as special cases, and develop novel approaches to automatically find the nearly optimal filter with the aim of reducing the costs of filtering and verification simultaneously.

4 General Gram Filter

In this section, we propose a general gram filter to resolve inadequacies in the existing framework. Before that, a lemma on which existing filters are all based is given first. The basic idea of Lemma 1 is that enough grams can prevent all the grams from being destroyed by edit operations so that the survived grams can be used to obtain the similar strings.

Lemma 1. *Let $S_q(s_1)$ be segment set (q-gram or q-chunk) for string s_1 and $G_q(s_2)$ be q-gram set for string s_2. Consider any subset $\hat{S}_q(s_1)$ of $S_q(s_1)$ of size $D + LB$, where D represents for the maximal number of grams destroyed by τ edit operations and $1 \leq LB \leq |S_q(s_1)| - D$. If $|S_q(s_1) \cap G_q(s_2)| \geq |S_q(s_1)| - D$, $|\hat{S}_q(s_1) \cap G_q(s_2)| \geq LB$. The contraposition of this statement can be used as the condition to remove dissimilar strings safely.*

Proof. Assuming $|\hat{S}_q(s_1) \cap G_q(s_2)| = m < LB$, then $\hat{S}_q(s_1)$ contains $D + LB - m$ grams that never appear in $G_q(s_2)$. $|S_q(s_1) \cap G_q(s_2)|$ will be at most $|S_q(s_1)| - D - LB + m$, which is less than $|S_q(s_1)| - D$. That conflicts with the given condition. Combine statement "if $|S_q(s_1) \cap G_q(s_2)| < |S_q(s_1)| - D$ s_1 and s_2 must be dissimilar" with statement "if $|\hat{S}_q(s_1) \cap G_q(s_2)| < LB$, then $|S_q(s_1) \cap G_q(s_2)| < |S_q(s_1)| - D$", dissimilar strings can be detected. □

Lemma 1 formally describes any subset whose size is no less than $D + 1$ can be used as query-gram set. Based on this lemma, general gram filter is defined as follows.

Definition 4 (General Gram Filter). *By using any subset $\hat{G}_q(s_1)$ whose size is $D + LB$ of query string q-gram set $G_q(s_1)$, $\hat{G}_q(s_1)$ can be used as query-gram set of a gram filter f in the candidate generation phase. Gram filter f is an instance of general gram filter. The corresponding lower bound is LB. Denote this query-gram set as $G^j_{LB}(s_1)$, where the j represents different combination grams for a given LB.*

In the above definition, f is variable. And only the query-gram set is fixed, the lower bound LB can be decided. To show general gram filter results in complete answers, we give the proof of completeness.

Completeness. Assuming s_1 is query and s_2 is string in database, based on the pigeonhole principle, since τ edit operations can at most destroy D grams of s_1, any $D + LB$ grams will guarantee at least LB gram of s_1 survive. The survived grams can be used to access s_2 through inverted index. s_2 occurs in at least LB lists, which meet the condition reserved by filter.

The value of D correlates with the chosen subset. Non-overlap subset can make D minimal [6], in which $D = \tau$.[5] The value of LB will be decided automatically during the process of finding efficient gram filter. The exact value affects the efficiency. Note that for a given LB ,there are $C^{LB}_{|G_q(s_1)|}$ different gram combinations. Prefix filters and counter filters can be seen as the special case by setting $LB = 1$ and $LB = |S_q(s_1)| - D$ respectively.

For example, consider a query string $abcdefg$ and threshold $\tau = 1$. $G_2(abcdefg)$ is $\{ab, bc, cd, de, ef, fg\}$. Let $LB = 1$, since an edit operation affects at most 2 grams, then any 3 grams in $G_2(abcdefg)$ can be used as the filter grams. The first $2 * 1 + 1 = 3$ grams in global order is the case of prefix filter. Consider $\tau = 2$ and $LB = 2$, the general gram filter size $2 * 2 + 2 = 6$ is equal to $|G_2(abcdefg)|$ that means all the grams are used which is the same as counter filter.

The number of general gram filters for a given query is huge. So a question is *which gram filter is the best filter?* In order to answer this question, we conduct an analysis on the performance of similarity query process in the sequel.

Theoretical analysis. The query process consists of candidates generation and verification.

- *Candidates generation*. Let $I(g)$ be the element set containing all the strings in inverted list of entry g. For each gram $g \in G_{LB}(s_1)$, existing framework needs to scan all elements in $I(g)$, the total cost is $\sum_{g \in G^j_{LB}(s_1)} |I(g)|$.
- *Verification*. Let $\mathbf{C}^j_{LB}(s_1)$ denote the candidate set of query s_1 which consists of strings that appear in at least LB lists of elements in $G^j_{LB}(s_1)$. And $cost_v(c)$ denotes the average cost to verify an element $c \in \mathbf{C}^j_{LB}(s_1)$. The total cost is $\sum_{c \in \mathbf{C}^j_{LB}(s_1)} cost_v(c)$.

By summing up the above two costs, we obtain the total cost of filter-and-verification framework using query-gram set $G_{LB}(s_1)$, i.e.,

$$\Theta^j_{LB} = \sum_{g \in G^j_{LB}(s_1)} |I(g)| + \sum_{c \in \mathbf{C}^j_{LB}(s_1)} cost_v(c). \quad (1)$$

5. Grams in non-overlap set will share no position among them. Since an edit operation can occur in at most single non-overlap gram, τ operations result in τ grams destruction.

The total cost of query process is dependent with the number of strings to be accessed and the number of candidates to be verified, both of which are affected by the grams chosen in the candidates generation step. Actually, for each LB there are $\binom{LB+D}{|G_q(s_1)|}$ query-gram sets as the candidate filters to be chosen. Let $g_i \in G_q(s_1)$. The objective function to optimize the choice of query-gram set is:

$$\arg \min_{g_i \in G_q(s_1), LB} \sum_{g_i \in G_q(s_1)} |I(g)| + \sum_{c \in \mathbf{C}^j_{LB}(s_1)} cost_v(c), \quad (2)$$

where:

$$\begin{cases} 1 \le j \le \binom{LB+D}{|G_q(s_1)|} \\ \sum_{g_i \in G_q(s_1)} g_i = LB \\ g_i = 0, 1. \end{cases} \quad (3)$$

$g_i = 1$ means gram g_i is chosen, otherwise $g_i = 0$.

Lemma 2. *Optimizing the objective function (2) is an NP-hard.*

Proof. We consider the special case of the problem, where $cost_v(c) \gg \sum_{g \in G^j_{LB}(s_1)} |I(g)|$, which means the verification cost dominates the total cost. If the special case is NP-hard, so is original problem. This problem can be stated as follows.

Special case problem. Given N inverted lists, corresponding to the query's q-gram set, and a constant D, pick up $D + LB$ grams that make the candidate set minimum, where a string will be a candidate if it appears in at least LB chosen inverted lists.

We will reduce the *minimum set-cover problem* to our special case problem. Let S be the solution of minimum set-cover problem. Before the proof, we introduce a lemma which will be used in reducing process. □

Lemma 3. $\bigcap_{s \in S} \bar{s} = \emptyset$, *where the* $\bar{s}$ *denotes the complement of set* s.

Proof of Lemma 3. Assuming the $\bigcap_{s \in S} \bar{s} = \delta \neq \emptyset$, then element in δ will never be covered by the S, which is a conflict with the fact that *S is a solution*.

For convenience to prove NP-hard, we can treat each inverted list as a set naturally. Constructing a solution O to a minimum set-cover problem that consists of LB sets. And let all the complements of sets in the minimum set-cover problem be the sets in our problem. Setting $D = 0$, then all the complements of set $o_i \in O (1 \le i \le LB)$ constitute a solution to our special problem. The reason is the intersection among these sets is empty (Lemma 3) and no candidate set size can be less than 0. That meets the minimal number of candidates, so this solution is correct to our special problem. We can also use this solution to solve minimal set-cover problem. □

Lemma 2 implies that getting the optimal query-gram set efficiently during the query process is impossible. Although this cost model can not be directly used to select grams, it enlightens us to develop heuristics. In the sequel, we will first develop a framework to choose the grams quickly, avoiding searching in enormous combinations of grams. Then, under the new framework, we propose a new criterion to help choose high quality grams, which proximately reach the goal of cost model.

5 Gram Set Selection Framework

Enumeration of all the possible gram combinations is time-consuming. In order to avoid searching in enormous combinations of grams, we propose a gram-set-selection framework to select gram one by one. Although different approaches can be proposed in this framework, two key problems that are critical to the overall performance need to be addressed: *how to measure the gain of adding a new gram in query-gram set* and *when to stop the selection process*. Inspired by the minimum set cover problem, in this section we propose a greedy algorithm as a baseline method to address the above problems.

5.1 Choose-and-Extend Framework

In Algorithm 1, we develop gram set selection framework to deal with similarity search. First, $D + 1$ grams are selected as base query-gram set (Line 1). Then new gram is added in query-gram set one by one until no benefit can be obtained (Lines 2-5). At last, all the candidates surviving at filter step will be verified (Line 6).

Algorithm 1. Choose-and-Extend Framework

Input: The q-gram set of query Q, $G(Q)$;
The number of grams destroyed by τ edit operations, D;
The inverted lists, I;
Output: The similar string set Re according to query

1: Choose $D + 1$ grams as the base query-gram set and generate the candidate set C_1;
2: **for** $i = 2; i <= |G(Q)| - D; ++i$ **do**
3: **if** *No benefits can be obtained from adding new grams* **then**
4: break;
5: Add g_i with largest benefit into the query-gram set and merge its inverted list with C_{i-1} to generate the candidate set C_i;
6: Verify the $C_{finally}$ set to generate the result set Re;
7: **return** Re;

5.2 Greedy Algorithm

Set cover problem can be approximately solved by using greedy algorithm. In this section, we propose a similar greedy algorithm. Since adding a gram is to sacrifice the accessing cost and benefit from verification cost, the greedy strategy is to find grams one by one until no benefit obtained. In order to measure the benefit, a ratio between the reduction cost in the size of the candidate set and new gram list length is used to quantify the gain of adding a new gram.

5.2.1 Effect on Cost Model by Adding Gram in Query-Gram Set

Assuming a base query-gram set is obtained, which contains $D + 1$ grams and common gram lower bound is 1. Let C_i denote the candidate set whose common gram lower bound is i and $C_i^=$ denote the subset, whose elements'

frequency is equal to the common gram lower bound, of C_i. Based on the total cost in Equation (1), absorbing new gram g_n may increase the access time but can reduce the candidate size that saves the verification time. Let B denote the reduction in the size of the candidate set and $I(g_n)$ denote new added gram g_n in inverted list. The $Gain$ of a newly added gram is evaluated by the following equation:

$$Gain(g_n) = |B| \cdot cost_v(Q) - |I(g_n)|, \tag{4}$$

where the $cost_v(Q)$ represents the average verification time between a query string and a candidate. And it can be estimated by real cost of calculating edit distance between query string and itself. $|I(g_n)|$ refers to accessed list length. $Gain$ is the difference between benefit of candidates reduction and cost of accessing inverted list. If the $Gain$ is larger than 0, it means this gram can be added to query-gram set, since it bring more benefit than cost. However, to directly compute this gain should access all the elements in the lists, which is time-consuming and not appropriate. Besides, it is unnecessary to access the inverted lists before the corresponding gram is selected as element of query-gram set. This calculation need a more efficient implementation.

Lemma 4. $|B| = \frac{|C_i^=|}{1 + J_S(C_i^=, I(g_n))} - \frac{|I(g_n)|}{\frac{1}{J_S(C_i^=, I(g_n))} + 1}$.

Proof. By the set exclusion principle, we have

$$|C_i^= \cup I(g_n)| = |C_i^=| + |I(g_n)| - |C_i^= \cap I(g_n)|, \tag{5}$$

dividing both side of Equation (5) by $|C_i^= \cup I(g_n)|$ and using

$$J_S(C_i^=, I(g_n)) = \frac{|C_i^= \cap I(g_n)|}{|C_i^= \cup I(g_n)|}, \tag{6}$$

we can obtain

$$|C_i^= \cup I(g_n)| = \frac{|C_i^=| + |I(g_n)|}{1 + J_S(C_i^=, I(g_n))}. \tag{7}$$

Since the reduced candidate set is:

$$B = |C_i^=| - |C_i^= \cup I(g_n)| \cdot J_S(C_i^=, I(g_n)), \tag{8}$$

by using Equations (4), (7) and (8), Lemma 4 can be obtained. □

We can make an observations from Lemma 4. B can be calculated by using $|C_i^=|$, $|I(g_n)|$ and $J_S(C_i^=, I(g_n))$. $|C_i^=|$ and $|I(g_n)|$ can be easily gotten so that we just need to calculate the $J_S(C_i^=, I(g_n))$.

By generating the Min-Hash [11] signature for $C_i^=$ and $I(g_n)$, $J_S(C_i^=, I(g_n))$ can be estimated by Min-Hash signature, which avoids access of inverted list. In an inverted list, all the strings in different length are grouped into different parts. And in each group, the strings are in the same length. Each group is given a Min-Hash signature. Take Fig. 4 as an example. In inverted list of gram ab, string ids in group 1-8 are in the same length and given signature "4, 0". Since there is no string id in the group 9-16, the signature is "null, null". After using length filter, the cost of estimating B is $O(\tau * n * |C_i^=|)$, in which h is the number of hash function. In our experiments, we only use five hash functions. To reduce the maintenance cost, we can use v-signature introduced in next section.

Since there is no candidate before base query-gram set is decided, the benefit for gram adding in base query-gram set is different. Recall, our goal is to choose grams that generate less candidates. The basic idea is gram lists having low overlap between each other result in small candidate set. The benefit can be measure through the similarity between selected merged gram lists and new gram list, which just replaces C_1 in Lemma 4 with $MergeList$, where the $MergeList$ represents merge list of selected gram lists. Two lists are merged into a single list by union their elements. To distinguish it from $Gain(g_n)$, we denote it as $Benefit(g_n)$.

5.2.2 Algorithm

The greedy algorithm revises framework in Algorithm 1 and is illustrated in Algorithm 2. SelectGramByBenefit() is to select gram with the largest $Benefit(g_n)$ in the left q-gram set. After the base query-gram set is fixed, new grams are appended until the largest $Gain(g_n)$ has no benefit, which is check in CheckLargestGain(). The worst cost of greedy algorithm is $O(n^2)$, where n is the number of query's q-grams.

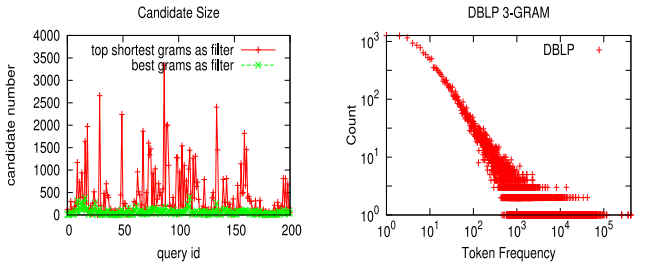
Algorithm 2. Greedy Algorithm

1: Add initial step "Add gram list of the smallest length in initial query-gram set";
2: Change Line 1 in Algorithm 1 to SelectGramByBenefit();
3: Change Line 3 in Algorithm 1 to CheckLargestGain();
4: Use $Gain(g_n)$ as the measure of gain in Line 5 in Algorithm 1;

Greedy is not good. The greedy algorithm can be easily trapped within local optimum result, since the first selected gram contains few information about candidate set. Moreover, $O(n^2)$ is expensive for online processing, where n is the number of grams of the query. In addition, the size of $C_i^=$ is also not small in the beginning, which can cause extra cost in filter step. In the next section, we propose a rank based method with two-step stop condition, which can handle these issues.

6 A Gram-Ranked Based Method

Gram set selection framework can reduce the cost of selection and the gain can help decide when to stop selection. Nevertheless, we still need more efficient and effective strategies to find high quality grams. In this section, we improve this framework by introducing a ranking method for better grams selection and a two-step stop condition with less extra checking cost. Since the string elements shared among gram lists decide the candidate size, we should preserve the comparison among gram lists to find lists with small intersection. Besides, comparison is also necessary to reflect the relative value of list length among a group of gram lists. Therefore, comparison sort is an appropriate method to help find the grams, achieving the goal of the cost function. And the time cost is just $O(n \log n)$. In this section, we also propose a new



(a) Candidate Size Distribution, DBLP

(b) Gram Frequency Distribution in DBLP

Fig. 3. Statistic in DBLP.

perspective to quantify the quality of grams for ranking. Compared with existing method, our new criterion can measure both access cost and filter ability of grams.

6.1 Problem of Existing Approaches

All approaches based on the prefix filter need a global ordering for the grams. The state-of-art method is to sort all the grams based on the *inverse document frequency (IDF)* of a gram. Let N denotes the number of strings in the string collection and n denotes the number of strings in a given gram's inverted list. Then the *idf* is $\frac{N}{n}$. Since N is a constant when the string collection is given, the idf is just dependent with the length of the whole inverted list. A global ascending order based on the length is used to help choose the top $D + LB$ grams. The intuition of this approach lies in that shorter lists result in less access time, so it mainly focuses on optimizing the cost in candidates generation phrase.[6]

Fig. 3a is the candidate size distribution for 200 randomly chosen queries on DBLP data set, in which threshold $\tau = 5$ and $LB = 2$ are set for all the queries. We compare the grams selected based on *idf* and the optimal grams which are found by brute force method. Optimal grams can significantly reduce the candidate size, but the gram chosen based on the length of list will easily result in a large candidate set. Since the time cost for evaluating the real edit distance is expensive, less candidates will bring more benefit than the extra list access cost. What's more, a global statistical length doesn't represent the real length used in query process after apply length filter, leaving a chance to reduce access cost.

6.2 Motivating Example

We still use the inverted list shown in Fig. 2 as the example. The second element of ef is 4 now. Suppose a query $Q =$ "$abcdefg$" generate grams $\{ab, bc, cd, de, ef, fg\}$, and $D = 1$, $LB = 2$. For the traditional prefix filter, query-gram set $\{ab, bc, cd\}$ will be chosen as the query-gram set. Then the candidates are $\{1, 3, 7, 9, 11\}$, since all of them have at least two occurrences in the inverted list. However, if we choose another three grams $\{ab, ef, fg\}$ as the query-gram set, the candidates are $\{1, 7\}$, where the number of candidates is reduced due to the small list overlap among the lists associated with the query grams. Although the operation time increases for extra three elements, the candidate size is reduced by more than a half, by which we can save $3 * (\tau + 1) * |Q|) = 3 * 2 * 7 = 42$ operations in the verification step (We use verification method proposed in [29]).

We observe through analysis based on real dataset that short gram inverted lists dominate the gram set of a string collection. Fig. 3b shows that 3-gram lists length in DBLP follows a power law distribution, which means the majority of gram lists is not very long. There is potential to select longer lists without sacrificing a lot of access cost, even though they are not optimized. If we can choose the a little longer gram lists that just generate a few candidates, the total cost is expected to decrease. Inspired by this observation, we may improve the quality of the constructed prefix by proposing a new ranking criterion for grams based on the potential number of candidates that it may generate. However, we are faced with several challenges for developing this new criterion:

- The number of candidates is related to the list overlap among several grams. But the number of combinations of grams is huge, rendering the way to examine all the combinations infeasible.
- We need to calculate the overlap between gram lists efficiently. The overlap between two sets can be estimated by using similarity-based estimation methods. But it is not straightforward to extend them to multi-grams, since the accumulated errors will affect the effectiveness of estimation.
- After all, the list overlap is a measure based on multiple grams' list, but the rank is a measure for individual gram.

6.3 Discrimination of Grams

Considering these issues, we propose a quantification on individual gram which involves the information among grams to measure the filter ability of grams. In order to deal with the efficiency, Min-Hash signature is used as the summary for each gram list.

6.3.1 Definition

List overlap is a measure on multiple gram lists, which can only be calculated after the grams are selected and fixed. In order to quantify each individual gram on how it will contribute to the list overlap, we develop the concept of *discrimination* of gram as follows.

Definition 5. *Given the gram set G and the corresponding gram inverted list set I, for each gram $g_i \in G$, the discrimination of g_i on G is defined as*

$$\sum_{s_j \in I(g_i)} Frequency_I(s_j) - 1, \tag{9}$$

where s_j is a string element in the inverted list $I(g_i)$, and $Frequency_I(s_j)$ is the occurrence frequency of s_j in the inverted list set I.

The rationale of using discrimination to choose gram is that gram lists consisting of less frequent elements usually have higher filtering power. We reduce the frequency by 1 in the above equation to eliminate the effect of list length. Since longer list with more elements also results in high value discrimination, it is hard to tell whether discrimination come

6. The global shortest is not the real shortest after the length filter.

from overlap information. Taking Fig. 2 as an example, the discrimination of gram fg is calculated by $(6-1)(s_1)+(1-1)(s_2)+(3-1)(s_7)+(1-1)(s_8)+(1-1)(s_{13})+(1-1)(s_{14})=7$, where $(6-1)(s_1)$ means there are 5 s_1 in these lists except the one contain in fg. All the *discrimination* of this gram set are $10, 10, 12, 8, 9, 7$ from top to bottom. Obviously, gram fg is the most discriminative which means it has very small list overlap with the other grams and thus higher selectivity if added into the query-gram set.

However, it is not feasible to calculate the true discrimination value for each gram, since it needs the access of all the elements in the inverted lists. It conflicts with the goal to avoid access. Next we propose to use the Min-Hash signature for efficient estimation for the discrimination.

6.3.2 Relative Discrimination

Calculate real discrimination need to access inverted list, which is time-consuming. Since this process is extra to query process, we should consume time as small as possible.

Recall that discrimination is a relative quantity, the relation among grams is key. To model the relation and reduce the computation, we can use Min-Hash signature. The intuition of using Min-Hash to estimate discrimination is that Min-Hash is a similarity-preserving signature, so we can use it to calculate the intersection among different inverted lists. Since the discrimination is to reflect the difference of a list towards a set of lists, the less intersection a list can share with other lists, the more discriminative it is. So we use the Min-Hash to estimate discrimination. The calculation of relative discrimination is similar to real discrimination while it counts elements of Min-Hash signature.

For example, consider inverted index in Fig. 2, by using teo hash functions $h_1(x)=(x+3) \bmod 11$ and $h_2(x)=(3x+2) \bmod 11$, we can generate Min-Hash signature in Fig. 4. Assuming strings in id range [1], [2], [3], [4], [5], [6], [7], [8] and [9], [10], [11], [12], [13], [14], [15], [16] are the same length respectively, there are two signatures for each gram list. Relative discrimination for each range is calculated first and then merged into an integrated value. To calculate the *relative discrimination* of gram fg, we count the frequency of Min-Hash values that occur in this gram set, i.e., $Frequency_I(0)=4$ and $Frequency_I(1)=1$ for the first range and $Frequency_I(5)=1$ and $Frequency_I(0)=1$ for the second one. Then the integrated relative discrimination of gram fg is calculated as $(4-1+1-1)/2+(1-1+1-1)/2=1.5$. The relative discrimination of other grams is also showed in column **1-sig DIS** of Fig. 4. Recall the query Q = "$abcdefg$", which generates grams $\{ab, bc, cd, de, ef, fg\}$, $D=1$ and $LB=2$. If we choose the 3 grams with least relative discrimination, $\{de, ef, fg\}$, the candidate set is $\{1, 4, 10\}$, which is smaller than $\{1, 3, 7, 9, 11\}$ which is the result of using $\{ab, bc, cd\}$.

The purpose of using Min-Hash is to approximate the intersection size of large gram lists with fewer hash values. Therefore the number of hash functions is a tradeoff between time cost and accuracy: the more hash function, more accurate discrimination values but higher time cost, and vice versa. Theoretically there has proved to be an expected approximation error of $O(\frac{1}{\sqrt{k}})$, where k is the number of hash function used. It implies that, in order to achieve an expected error ϵ, we need to use $O(\frac{1}{\epsilon^2})$ hash functions. We conduct more empirical study in Section 7.2 to evaluate the effect of the number of hash functions.

	1-sig			2-sig	
gram	$1-8$	$9-16$	**1-sig DIS**	$1-16$	**2-sig DIS**
ab	4,0	null,null	3.5	4,1	2
bc	4,0	1,2	5	4,3	3
cd	4,0	1,2	5	4,3	3
de	4,3	2,3	3.5	2,0	1
ef	4,3	1,6	3.5	3,0	1
fg	0,1	5,0	1.5	0,5	0

Fig. 4. Signature of inverted index.

6.4 V-Signature

With the help of Min-Hash signature, we can estimate the discrimination by avoiding accessing the inverted lists directly. However, even though Min-Hash signature can be pre-computed, this technology still introduces extra hash values to access. In order to minimize the cost of estimating discrimination, the number of hash values should be as less as possible.

Instead of just grouping strings of certain length into a range, we can build multi-version signature for each list. We call this method **v-signature**. After grouping the strings by their length, a set of groups, denoted by $(g_1, \ldots, g_i, \ldots, g_n)$, can obtained, where g_i denote the group whose strings' length are i. Similar to the definition of q-gram, v-signature is to build a Min-Hash signature for strings in a range which consists of v groups starting from i $(i \leq n-v+1)$. If a query come, we choose a non-overlap combination with the least number from these multi-version signature as the query-signature. For example, in Fig. 4, 1-signature is the case of building a Min-Hash signature for each group. To build 2-signature, we combine strings in [1], [2], [3], [4], [5], [6], [7], [8] and [9], [10], [11], [12], [13], [14], [15], [16]. Still using the same hash functions mentioned in last section, we can calculate 2-signature for [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]. Among 1-signature and 2-signature, only using a signature for [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16] from 2-signature can result in the least number of combination. Two-signature can also help us pick up gram like 1-signature, which only needs access half of the hash values. For different τ, we use different version of v-signature so that the access time of Min-Hash signature can be reduced. The complexity of using v-signature to estimate the relative discrimination is $O(\tau * n * |Q|)$, where the n is the number of hash functions and $|Q|$ is the length of query string.

6.5 Synthetic Criterion

Although discrimination can help reducing candidate set, it still ignores filtering cost and long list may be introduced. Gram list length should be considered as well. Besides, global list length information can not predict the real length

used in query process, since length filter will affect the real used strings in list. Consider query string s, length filter will ignore all the strings whose length exceed the range $[|s|-\tau, |s|+\tau]$. To solve these problems, a synthetic rank is illustrated in Algorithm 3. First, the normalization for list length and discrimination are calculated respectively (Lines 1-2). $U(g)$ stands for the discrimination of gram g, and $|I_G^\tau(g)|$ stands for the strings that survive in length filter under the edit distance τ. The discrimination and list length are combined into the total rank of gram after normalization (Lines 3-4), which is sorted in ascending order (Line 5). The time complexity of this algorithm is $O(N + N \cdot \log N)$, in which the N is the number of grams.

Recall the calculation of the *Gain* in Equation (4) and Lemma 4, another observation is $Gain \propto -|I(g_n)|$ and $Gain \propto \frac{1}{J_S(C_i, I(g_n))}$. It suggests gram with smaller length and lower list overlap with others can have large *Gain*. This is similar to synthetic criterion, which aims to rank shorter and more discriminative grams in the front of order. Inspired by this observation, we can simply select grams, which will be used in next section, in turn.

Algorithm 3. RankGram()

Input:
The discrimination set of gram set, U;
The gram set corresponding inverted lists of $G(Q)$, I_G;
The edit distance τ;
1: **for** *each* $g \in G(Q)$ **do**
2: $Norm_{discrimination} += U(g)$;
$Norm_{length} += |I_G^\tau(g)|$;
3: **for** *each* $g \in G(Q)$ **do**
4: $rank(g) = (1-\lambda) \cdot \frac{U(g)}{Norm_{discrimination}} + \lambda \cdot \frac{|I_G^\tau(g)|}{Norm_{length}}$;
5: Sort grams based on $rank$;

6.6 Gram-Ranked Based Method with Two-Step Stop Condition

To this end, we analyze the rationale to use synthetic criterion select high quality query-gram set for a given query and quantify the gain of new appended gram.

Based on analysis above, we develop gram set selection framework to deal with similarity search in Algorithm 4. First, discrimination of grams is calculated (Line 1). Then gram is sorted based on synthetic criterion in $RankGram(G(Q), \tau)$(Line 2). In order to choose high quality query-gram set, a base query-gram set consisting of top minimal $D+1$ grams are first chosen (Line 3). New grams in the ranked list are added in query-gram set one by one until gram gain is less than 0 (Lines 4-8). In order to reduce the cost of maintaining signature of $C_i^=$, we propose a light-weight stop condition in Line 5. This condition is the lower bound of the gain. By simply assuming that all the elements in the new added gram's list are shared by $C_i^=$, we can get the minimal gain $|C_i^=| - |I_i|) * cost_v(Q) - |I_i|$. After that condition, the size of $C_i^=$ is small and we can use the $Gain(g_i)$ to calculate a more precise gain efficiently in Line 6. Since gain is also calculate by length range so that the algorithm stops. At last, all the candidates surviving at filter step will be verified (Line 9).

Finally it is worth to note that many existing advanced filters and optimization, such as those mentioned in [4], [6], can still be used under our framework.

Algorithm 4. Gram-Ranked Based Framework

Input:
The q-gram set of query Q, $G(Q)$;
The number of grams destroyed by τ edit operations, D;
The Min-Hash signature set of gram lists, S;
The inverted lists, I;
Output:
The similar string set Re according to query
1: CalculateDiscriminationOfGrams(G(Q));
2: RankGram(G(Q),τ);
3: Choose the top $D+1$ grams as the base query-gram set and generate the candidate set C_1;
4: **for** $i = D+2; i < |G(Q)|; ++i$ **do**
5: **if** $(|C_i^=| - |I_i|) * cost_v(Q) - |I_i| < 0$ **then**
6: **if** $Gain(g_i) < 0$ **then**
7: break;
8: Add ith gram g_i in the ranked lists into the query-gram set and merge its inverted list with $C_i - 1$ to generate the candidate set C_i;
9: Verify the $C_{finally}$ set to generate the result set Re;
10: **return** Re;

7 Experimental Evaluation

In this section, we present our experimental results of the proposed techniques. The effect of different factors on the performance of our framework is evaluated.

7.1 Experimental Setup

We implement our method to support similarity search, and compare it with other state-of-the-art algorithms. All the experiments are conducted in the main memory. In particular, Adaptive-search [7] is a prefix-filter based method. It extends the prefix length of the classic prefix filter. We implement this method without the delta inverted index, since the threshold τ is a part of query and cannot be known as a priori in the similarity search problem. ChunkGram [6] is also a prefix-filter based method with advanced filters. We also use advanced filter in Pass-Join [29] to improve ChunkGram. We use inverse document frequency as the global order when necessary. Although there are many other methods such as VGram [8], PartEnum [16], B^{ed}-Tree [17], Trie-join [23], and ED-Join [4], prior works [6], [10], [7] have shown that none of them can outperform the above-mentioned algorithms.

We adopt three commonly-used, real data sets to evaluate the performance in our experiments.

- **DBLP** strings are obtained from the DBLP Bibliography.[7] Each string is a concatenation of authors' names and article title.
- **Uniref** is the Uniref50 protein sequence data from UniProt project.[8] Each protein sequence is treated as a string.

7. http://www.informatik.uni-trier.de/~ley/db
8. http://www.uniprot.org/

Data Sets	*avg_len*	*max_len*	*min_len*	Sizes
Uniref50	406	4017	120	878,148
TREC	240	779	100	417,698
DBLP	105	1626	28	650,207
DBLP-author	39	100	16	650,207

Fig. 5. Statistics of datasets.

- **TREC** is the data set used for TREC-9 Filtering Track.[9] We remove the strings with less than 100 letters.

Fig. 5 provides more details about the data sets. Queries are randomly selected from the corresponding data set.

Parameter. Parameters q and λ should be set. To make q-gram-based method have pruning power, the threshold cannot exceed $\lfloor \frac{|s|-q}{q} \rfloor$. Since we can't predicate the threshold in advance, we set the $q = 3$ to support large edit distance threshold. We specify λ as follows: if $\tau < 5$, $\lambda = 0.2$; if $5 \leq \tau < 10$, $\lambda = 0.5$; if $\tau \geq 10$, $\lambda = 0.8$. When τ is small, the candidate is small, then the filter cost dominant the total cost; with the increase of the τ, the candidate increase exponentially, hence we should focus on reducing the verify cost.

Implementation of v-signature. In order to save the storage of Min-Hash signature, we only use four version of v-signature, where $v \in \{1, 2, 4, 8\}$. To support different τ, we can use the combination of the four level v-signature. For example, the $\tau = 20$, we can use two signature in 8-signature and one signature in 4-signature. To minimize the estimation cost, the combination of the least number of signature is picked up.

All the algorithms were implemented in C++ and compiled by Microsoft Visual Studio 2010. All the experiments were run on a Windows 7 machine with an Intel Core 2 Duo T6570 2.10 GHz processor and 8 GB main memory.

7.2 Effect of the Number of Hash Functions

The Min-Hash signatures of inverted lists are generated by N hash functions. In fact, the number of hash functions is a trade-off. If N is too large, the estimation is more accurate, but the cost of estimation will increase. Otherwise, if the N is too small, the estimation cost decreases, but the accuracy will be low and result in large candidate size. We test both the accuracy and the efficiency. Since the relative discriminative just records the relative rank of gram, we use the NDCG to measure the distance that the relative discriminative rank deviate from the real discriminative rank. In the result, the NDCG increase with the number of hash functions. The reason is that the more hash function we use, the higher accuracy we obtain. Since this factor affects the whole query process, the efficiency is also measured on the query-gram set selection framework. We vary N under different query threshold τ. Fig. 6 shows the results. The correlation between the cost and the number of hash functions is different on the three datasets. For example, the number of hash functions do not influence the running time much on DBLP dataset, whereas the choice of five hash functions can lead to a better performance in general for the Uniref50 and TREC datasets. It implies the hash function number should not be either too small or too large. Based on the result, we set N in our experiment to be 5, in which the time cost is small and the NDCG is high.

9. http://trec.nist.gov/data/t9 filtering.html

7.3 Similarity Search Performance

In this section, we compare our query-gram set selection framework with the state-of-art methods for similarity search problem.

We compared our approach with Adaptive-Search and ChunkGram. Greedy algorithm is also illustrated.

- *Greedy*. Greedy algorithm described in Section 5.2 is implemented. V-signature is used to speed up similarity calculation. Length filter and position filter are used.
- *Adaptive-search*. We implement Adaptive-Search[7] with delta inverted index whose first part store 1-prefix, since delta inverted index should know the query threshold in advance. Adaptive-Search use the same inverted index like other approaches. The sort is based on the global list length. The sample ratio of Adaptive-Search is set to 1 percent, as recommended by [7].
- *ChunkGram*. We use the source code provided by the author of ChunkGram [6]. ChunkGram is a prefix filter based similarity search method.
- *CF-select*. We implement our method called chunk-based-filter with selection (CF-Select). The query string is partitioned into q-chunks. The sort is based on the hybrid function considering both list length and discrimination described in Section 6.5. Length filter and position filter are used.
- *GF-select*. Different from CF-Select, gram-based-filter with selection (GF-Select) partitions the query string into q-grams. The sort is also based on the hybrid function considering both list length and discrimination. Length filter and position filter are used.

Fig. 7 shows the average query cost, from which we made some conclusions: (1) CF-Select and GF-Select outperform other algorithms in general. GF-Select achieves better performance than CF-Select, when the query threshold is large in DBLP and TREC. We believe the reason is GF-Select can find higher quality query-gram set, since the chunk set is just a subset of gram set. However, in Uniref50 data set CF-Select is better than GF-Select. The causality is that small alphabet of protein sequence leads to small gram space and a protein sequence is usually very long so that each list will get high probability to sharing many strings with others. Grams in gram set are not better than that in chunk set significantly and gram set need more time in estimation step. (2) Greedy algorithm can achieve the similar performance to CF-Select and GF-Select when τ is large, due to the side effect of the first chosen gram disappears when most of grams are selected. However, in small τ case, it will spend more time on estimation. (3) Our approach is a little worse than Adaptive-Search in DBLP when τ is 4 (CF-Select/GF-Select/Adaptive-Search,1.53/1.65/1.24), due to DBLP set just produce a few candidates when τ is small. Adaptive-Search can save more time in filter step. But most of time, CF-Select and GF-Select are outperform Adaptive-Search since our method can use less gram than Adaptive-Search. (4) With query threshold

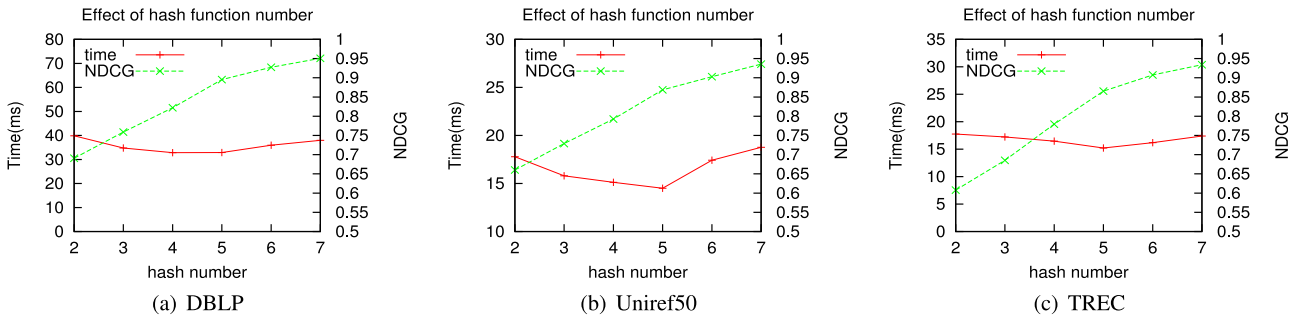


Fig. 6. Effectiveness of hash function.

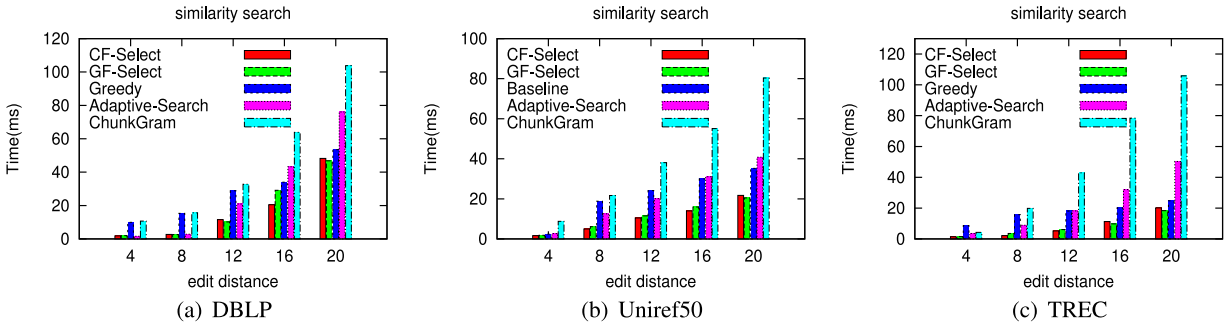


Fig. 7. Comparison of general gram filter with query-gram set selection similarity-search algorithms and existing methods.

increasing, our approach achieves the smallest marginal cost since it can remove more candidates. (5) Adaptive-Search turns to be the worst in the Uniref50 dataset while CF-Select and GF-Select perform quite stable. The reason is that long list will introduce much sampling cost with the increase of query threshold. But our approach will never suffer from this, since the estimation cost is linear with respect to a constant τ. (6) The performance of ChunkGram is not competitive, due to the 1-prefix scheme used in candidate generation, which will produce a large number of candidates.

Effect of query string length on efficiency. For different length of query string, we need to use different τ to achieve the same similarity. In order to investigate how the length of string can affect the efficiency, we compare our methods with the state-of-art methods under a fixed similarity. And we set τ to 20 percent of the query length. The experiment is conducted on data set DBLP-author name dataset. We calculate the average cost of randomly selected 300 query strings for each length and the result is illustrated in Fig. 8. When the query is short, the estimation cost is also reduced.

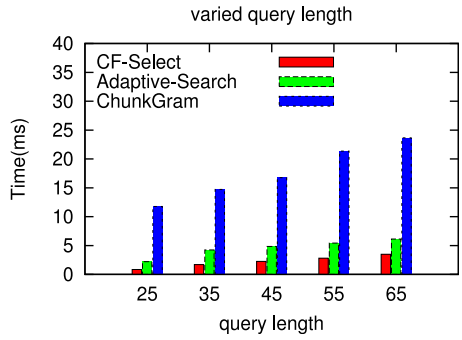


Fig. 8. Effect of the length of query on performance.

Our method can still outperform other methods, since we only sacrifice a little cost and save more cost for quickly finding small candidates.

7.4 Effectiveness of Using Query-Gram Set Selection

Adding grams properly can improve the query efficiency significantly. In order to verify the effectiveness of our query-gram set selection technique, we compare it with a fixed-length-extend prefix filter. A fixed-length-extend prefix filter is constructed by giving $D + r$ length prefix to the filter, where r is a constant. The hybrid ranking criterion considering both IDF and discrimination is applied to both methods. As shown in Fig. 9, we observe that the selection method always incurs less time cost than the fix-length prefix scheme. For example, on the DBLP data set, when the query threshold is 30, even the best scheme was 8-prefix scheme, its cost (204.12 ms) is still large than selection scheme (203.15 ms). Since all the hybrid criterion was used, the fix-length prefix scheme can also achieve good performance, if a proper prefix length is chosen. The results also shows for different query threshold τ the best prefix length changes differently. But when query threshold changes to 25, the best is 4-prefix scheme. By using our query-gram set selection method, the best scheme can always be found. However, when the query threshold is very small regarding the query length, query-gram set selection will cost extra time. For example, on the Uniref50 data set, when the query threshold is 10, 4-prefix scheme achieved the best performance, since it never take extra cost to test whether the prefix should be extended.

To explicitly show effectiveness, trends of average candidates and average accessed list elements with varying

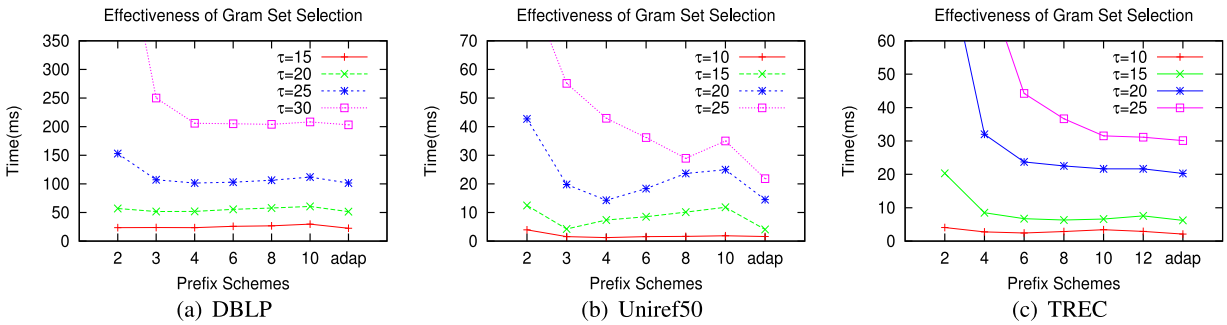


Fig. 9. Comparison of query-gram set selection scheme and fixed-length scheme.

adding extra grams for a single query is also shown in Fig. 10 under the $\tau = 10$. By selecting grams based on synthetic criterion, the candidates are reduced significantly and quickly. Even though extra elements are accessed, compared with advanced filter cost and verification cost to for each string in a large candidate, we can still benefit from this operation.

7.5 Effectiveness of Discrimination on Candidate

In this section, we evaluate the effectiveness of the proposed discrimination concept to reduce the number of candidates. If the candidate size cannot be significantly reduced by choosing grams based on the discrimination, it will be impossible for us to choose better gram filters. In this experiment, we choose the same number of grams based on discrimination and IDF respectively and compare their performance. The hybrid method considers both discrimination and IDF.

- *IDF*. Sort the grams of query by the gram list's global length in ascending order.
- *DIS*. Sort the grams of query by the gram discrimination in ascending order.
- *Hybrid*. Sort the grams of query by linear function that combines real used list length and discrimination.

We set the τ to 10, and vary k from 2 to 5, where k is the value of LB. Fig. 11 shows that the total candidate size decreases with the increase of k of 1,000 randomly selected queries. Besides, both DIS and Hybrid can generate much less candidates than IDF does, which demonstrates the discrimination can reduce the candidate size effectively. The hybrid approach always generates the least number of candidates.

7.6 Cost of Discrimination Estimate

In this experiment, we evaluate the extra cost of estimation. If the cost to estimation is expensive, this operation introduces more extra cost than the benefits to the query process. Fig. 12 shows the comparison results between the estimate cost of discrimination and the total query process cost. We can see that the estimation cost is much smaller (i.e., orders of magnitudes lower) than the total query cost. With the query threshold increasing, the query process cost increases significantly but the estimation cost remains quite stable. This means the discrimination estimation only incurs a little extra time, and verifies our assumption that the estimation cost is limited. For example, on the DBLP set, when query threshold is 25, the query process takes totally 89.76 ms while only 1.18 ms is used for the discrimination estimate.

	Uniref			TREC	
grams	elems	cands	grams	elems	cands
4.6	10085	3	6.7	27045	7
2	8672	1088	3	22783	1576
3	9268	197	5	25107	218
4	9872	43	7	28323	5
5	10539	2	9	33724	2

Fig. 10. Trend of candidate number and inverted list access.

8 Related Work

The problem solved in this paper is defined that: finding strings those similar to a given single query string from a string collection. We refer our problem as "string similarity search", which is widely studied in [1], [5], [6], [8], [9]. Please note we focus on processing exact similarity search query which is different from approaches introduced in [28].

Similarity join is to find all the similar pairs whose elements are from different sets. It can be solved in two different ways. One way is to generate signatures or data structures for set, then process based on set signatures is used to remove dissimilar pairs. Literatures [16], [23] belong to this aspect. [23] perform . Another method treat similarity join as a batch of similarity searches. All the existing approaches to process in this way can be classified into three categories. Fixed-length substring based method is the most common way. Query string is first segmented into q-grams then prefix filter [5] or counter filter [1] is used to prune irrelevant strings. Together with those elementary filters, advanced orthogonal filters [4] based on mismatching q-grams are proposed to further prune candidates. Qin's approach [6] proposes an improved approach based on prefix filtering. His main idea is to minimize the number of signatures by introducing chunk to gram matching. Pass-join [29] is also a prefix filter based method, which involves advanced filter using substring position matching information. Variable-length substring based method is another category. Different from fixed-length substring based methods, variable-length substring methods always calls for extra efforts in generating high quality variable-length substrings. VGRAM [8] [9] is proposed that generates less grams and reduce inverted list merging in filtering step. Tree-based method is from different perspective to handle this problem. Zhang et al. [17] uses a B^+-tree to store

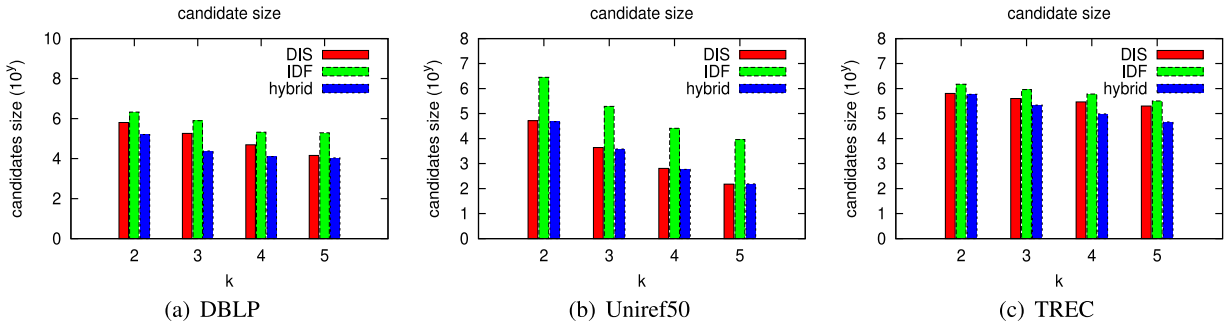


Fig. 11. Effectiveness of discrimination on size of candidates.

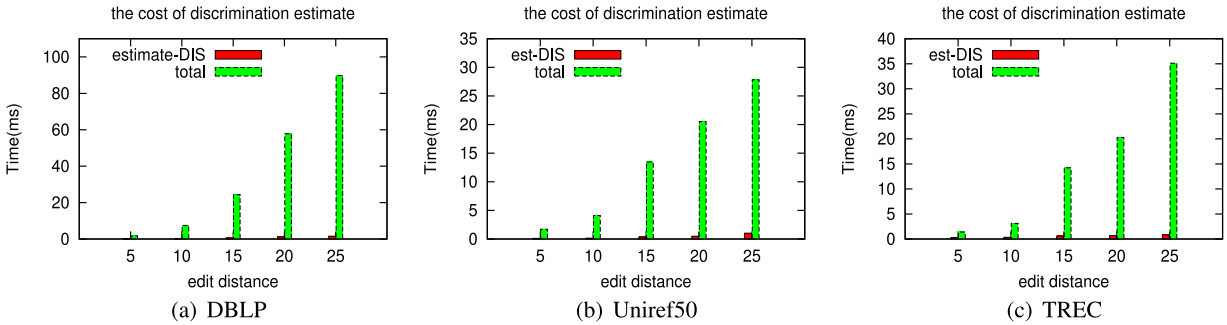


Fig. 12. Evaluating efficiency of estimate discrimination.

projected integers of strings to support edit distance similarity search efficiently. A main difference between similarity join and similarity search is that inverted index must be built before query process in similarity search, which means many optimizations and filters in similarity join by using the τ information can't be used in similarity search.

Our method can be used in fixed-length substring based and variable-length substring based approaches, and advanced filters in [6], [10], [29] can also cooperate with our filter. Different from previous studies, our approach focuses on optimizing the filter power by exploiting the inverted lists information, which can significantly enhance the performance of the candidates generation in existing frameworks, especially when the edit distance threshold increases. Although [4] proposes a technology to integrate prefix filter and counter filter and [31] develop a framework to better combine different types of filters, our work focus on using less filter and achieve better performance. Besides, it is the first attempt to maximize prefix filter's performance.

Different similarity or distance functions used for string similar searches have been studied, such as overlap, Jaccard similarity, cosine similarity, edit distance metrics, Bregman Divergence, and Earth Mover's Distance [1], [5], [13], [14], [15]. Edit distance is the most common way to measure the difference between two strings.

A variant, top-k string similarity search, is also studied in [30]. Another related problem, *Approximate string match*, refers to the problem of matching a string or sub-string to a given pattern in a text. There have also been a lot of studies on this problem [21], [24], and Navarro gives a detailed analysis on the existing approaches in his survey [12]. The selectivity estimation of similarity search and similarity join in [3], [25].

9 Conclusions

In this paper, we have studied the problem of efficient processing of string similarity search. A general gram filter is proposed, which is a novel generalization of existing filters. We first conduct a theoretical analysis for this filter model and prove that it is NP-hard problem to obtain optimal gram filter. Greedy algorithm is not good when τ is not large. We then devise an effective criterion to sort grams based on their potential capability to reduce candidate size, which is used to select better grams efficiently and effectively. Based on this, a new selection algorithm is also developed to construct the high quality query-gram set. Finally we conduct extensive experiments based on multiple real datasets and results demonstrate that our proposed framework can outperform the state-of-art methods significantly.

Acknowledgments

This work was supported by 973 project(No. 2010CB328106), NSFC grant (No. 61033007, 61170085 and 61021004), Shanghai Knowledge Service Platform Project No. ZF1213, and Shanghai Leading Academic Discipline Project grant (No. B412).

References

[1] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate string joins in a database (almost) for free," in *Proc. 27th Int. Conf. Very Large Data Bases*, 2001, pp. 491–500.

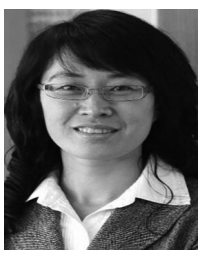
[2] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava, "Fast indexes and algorithms for set similarity selection queries," in *Proc. 24th Int. Conf. Data Eng.*, 2008, pp. 267–276.

[3] H. Lee, R. T. Ng, and K. Shim, "Similarity join size estimation using locality sensitive hashing," in *Proc. Int. Conf. Very Large Data Bases*, 2011, pp. 338–349.

[4] C. Xiao, W. Wang, and Xuemin Lin, "Ed-join: An efficient algorithm for similarity joins with edit distance constraints," in *Proc. Int. Conf. Very Large Data Bases*, 2008, pp. 933–944.
[5] S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," in *Proc. 22nd Int. Conf. Data Eng.*, 2006, pp. 795–825.
[6] J. Qin, W. Wang, Y. Lu, C. Xiao, and X. Lin, "Efficient exact edit similarity query processing with the asymmetric signature scheme," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 1033–1044.
[7] J. Wang, G. Li, and J. Feng, "Can we beat the prefix filtering? An adaptive framework for similarity join and search," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 85–96.
[8] C. Li, B. Wang, and X. Yang, "VGRAM: Improving performance of approximate queries on string collections using variable length grams," in *Proc. Int. Conf. Very Large Data Bases*, 2007, pp. 303–314.
[9] X. Yang, B. Wang, and C. Li, "Cost-based variable-length-gram selection for string collections to support approximate queries efficiently," in *Proc. Int. Conf. Very Large Data Bases*, 2008, pp. 353–364.
[10] C. Xiao, W. Wang, X. Lin, and J. X. Yu, "Efficient similarity joins for near duplicate detection," in *Proc. 17th Int. Conf. World Wide Web*, 2008, pp. 131–140.
[11] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. Compression Complexity Sequences*, 1997, pp. 21–29.
[12] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surveys*, vol. 33, no. 1, pp. 31–88, 2001.
[13] R. J. Bayardo, Y. Ma, and R. Srikant,, "Scaling up all pairs similarity search," in *Proc. Int. Conf. World Wide Web*, 2007, pp. 131–140.
[14] Z. Zhang, B. C. Ooi, S. Parthasarathy, and A. K. H. Tung, "Similarity search on Bregman divergence: Towards non-metric indexing," *VLDB Endowment*, vol. 2, no. 1, pp. 13–24, 2009.
[15] J. Xu, Z. Zhang, A. K. H. Tung, and G. Yu, "Efficient and effective similarity search over probabilistic data based on earth mover's distance," *VLDB Endowment*, vol. 3, no. 1, pp. 758–769, 2010.
[16] A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *Proc. Int. Conf. Very Large Data Bases*, 2006, pp. 918–929.
[17] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava, "Bed-tree: An all-purpose index structure for string similarity search based on edit distance," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 915–926.
[18] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2003, pp. 76–85.
[19] W. Wang, J. Qin, C. Xiao, X. Lin, and H. T. Shen, "VChunkJoin: An efficient algorithm for edit similarity joins," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 8, pp. 1916–1929, Aug. 2012.
[20] T. Bocek and B. Stiller, "Fast similarity search in large dictionaries," Dept. Inf., Univ. Zurich, Zurich, Switzerland, Tech. Rep. ifi-2007.02, 2007.
[21] W. Wang, C. Xiao, X. Lin, and C. Zhang, "Efficient approximate entity extraction with edit constraints," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 759–770.
[22] S. Chaudhuri and R. Kaushik, "Extending autocompletion to tolerate errors," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 707–718.
[23] J. Wang, J. Feng, and G. Li, "Trie-join: Efficient trie-based string similarity joins with edit-distance constraints," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 1219–1230, 2010.
[24] C. Sun, J. Naughton, and S. Barman, "Approximate string membership checking: A multiple filter, optimization-based approach," in *Proc. Int. Conf. Data Eng.*, 2012, pp. 882–893.
[25] A. Mazeika, M. H. Bohlen, N. Koudas, and D. Srivastava,, "Estimating the selectivity of approximate string queries," *ACM Trans. Database Syst.*, vol. 32, no. 2, p. 12, 2009.
[26] S. F. Altschul, W. Gish, W. C. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *J. Mol. Biol.*, vol. 215, no. 3, pp. 403–410, 1990.
[27] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Comput. Netw. ISDN Syst.*, vol. 29, nos. 8–13, pp. 1157–1166, 1997.
[28] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. 34th Annu. ACM Symp. Theory Comput.*, 2002, pp. 380–388.
[29] G. Li, D. Deng, J. Wang, and J. Feng, "Pass-join: A partitionbased method for similarity joins," *Proc. VLDB Endowment*, vol. 5, no. 3, pp. 253–264, 2012.
[30] D. Deng, G. Li, J. Feng, and W. Li, "Top-k string similarity search with edit-distance constraints," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 925–936.
[31] C. Li, J. Lu, and Y. Lu, "Efficient merging and filtering algorithms for approximate string searches," in *Proc. IEEE 24th Int. Conf. Data Eng.*, 2008, pp. 257–266.

Haoji Hu is currently working toward the master's degree in the Software Engineering Institute at East China Normal University. His research interests mainly include similarity query processing and semi-supervised learning.

Kai Zheng received the PhD degree in computer science from the University of Queensland in 2012. He is currently an Australia Research Council Discovery Early-Career Researcher Award fellow with the University of Queensland. His research interests include uncertain database, spatial-temporal query processing, and trajectory computing. He is a member of the IEEE.



Xiaoling Wang received the PhD degree from Southeastern University in 2003. She is currently a professor in the Software Engineering Institute at East China Normal University. Her research interests include XML data management and data management for data-intensive computing. She is a member of the IEEE.

Aoying Zhou received the PhD degree from Fudan University in 1993. He is currently a professor in the Software Engineering Institute at East Normal University of China. His research interests include web data management, data management for data-intensive computing, management of uncertain data, data mining and data streams, distributed storage, and P2P computing. He is a member of the IEEE.